# [430-555] Homework 4_2 : MVCC implementation

**Student Name: LE VAN DUC** **Student ID: 2016-27885**

**1. System configuration:**

**CPU**: Intel® Core™ i5-4460 CPU @ 3.20GHz × 4

**OS**: Ubuntu 14.04 LTS 64 bit

**RAM**: 8 GB

**HDD**: 1.1 TB

**2. C++ compiler configuration:**

*) g++: version 4.8

*) CXXFLAGS = -O2 -g -Wall -fmessage-length=0 -std=c++11 -lstdc++ -lsqlparser -L/usr/local/lib/ -I/home/duclv/work/sql-parser/src/ -I/usr/local/boost_1_61_0 –I/server

*) server code: use supplied server code and add –I/server on Makefile to compile.

*) client code: use supplied client code on folder client, compile by supplied Makefile.

*) Use boost library at: http://www.boost.org/ => compile config: -I/usr/local/boost_1_61_0

*) Use open source SQL parser at: https://github.com/hyrise/sql-parser => compile config: -L/usr/local/lib/ -I/home/duclv/work/sql-parser/src/

*) Use open source bit packing PackedArray (https://github.com/gpakosz/PackedArray): already added header and source file into my source code.

**3. Classes/files description:**

*** Core classes as in Homework 2:**

- **Table**: this class represents for a Table in main memory with columns.

- **ColumnBase**: base class for all columns.

- **Column**: this class represents for a column. Contains: Column's Dictionary, Data space vector, Version space vector, and Delta space (keeps values in Update operator).

- **Dictionary**: this class represents for the Dictionary and Delta space of Column.

*** Other classes and files:**

- **Transaction**: this class represents for Transactions of database. It keeps all transactions of database in a static vector (to share in total system). It also keeps list of **Active** Transactions (in processing transactions) and **Waiting** Transactions (in Write conflict transactions).

- **Garbage Collector**: this class represents for the garbage collecting process of database. It keeps a vector of recently updated row ids and run periodically as a Thread.

- **Folder client**: used to create a client to database.

- **Folder server**: used to create a server to receive requests from clients.

- **server_main.cc**: this file receives commands from client; process commands (INSERT, UPDATE, SCAN) and returns result to client. It also initializes the Garbage collecting thread.

- **App.cpp**: this file processes commands from server: INSERT, UPDATE, and SCAN.

**3. Implementation:**

**3.1. Insert operator:**

- Command format: **INSERT|attr1|attr2|attr3…** with attr1, attr2 …are values will insert into the columns of table Orders in order: o_orderkey, o_orderstatus, o_totalprice, o_comment. So, this command must have **at least 5 fields** (include INSERT).

- When insert into database, we just insert into **Data space only**, not insert in Version space because we want the inserted value will present immediately to users.

- The **insert Transaction** will be created new and get the start time stamp.

- The value inserted will be updated into each column's Dictionary and the dictionary encoding value will be added to vector of encoded value of column (can be bit packing). The Data space vector will consist of elements which point to encoded value vector and each element will keep addition 2 fields: **csn** = insert Transaction start time stamp; **version_flag** = default false because this element is just created new. See **figure 1** for more detail of this data structure:

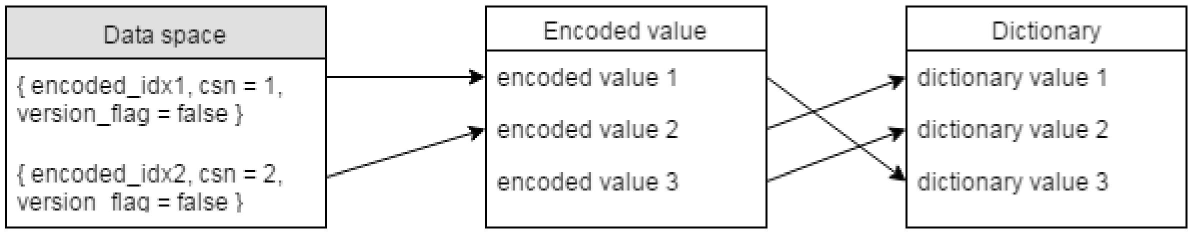


Figure 1: Column-store Data space structure

- After inserting successfully to Data space, the **Transaction will be committed**.

- File-function: **App.cpp - insertCommand; Column.h - insertDataVecValue**

## 3.2. Update operator (MVCC implementation):

- Command format: **UPDATE|rid|attr 1|attr 2…** with rid is the row id will be updated; attr1, attr2 … will be the values update to corresponding columns in order: o_orderkey, o_orderstatus, o_totalprice, o_comment. So, if we want to update which fields then we must supply their values in the command.

- When update a row id of table, we will **add to Version space and defer the update** to Data space for garbage collection.

- The **updating Transaction** will be created new and get start time stamp (startTs).

- To **avoid Write conflict** (that is 2 transactions update simultaneously on a row id) we will first check the transaction's **startTs** with updated row id's **csn**. If **startTs >= csn** then this Transaction can update to this row id; otherwise, it must be put in **Waiting transaction list** to update later.

- Also to avoid Write conflict, the updating row id's csn will be **set to maximum value** so that another Transaction cannot update it (because startTs will be always less than csn).

- The updated value to each column of this row id will be added to **Version space** with value updates to **Delta space** (to avoid continuously update Dictionary) and index from Delta space will be returned to version encoded vector. The Version space consists of elements that point to version encoded vector and has addition 2 fields: **csn**: Commit Sequence Number; **next**: pointer to next element. See **figure 2** for more detail of this data structure.

- Next, we will **update Hash table** by first checking on Hash table to find the previous version of this row id. If found an entry on Hash table with rid = row id then we got pointer to previous version on Version space. We point the next pointer of new created version to previous version and replace the previous version on Version space vector by new version => the index on Hash table will not change.

- If we do not find any entry on Hash table with rid = this row data's row id then we will add new version to Version space vector and **create a new entry on Hash table** as <rid, index on Version space vector>.

- Later, the Garbage collection will update Data space to newest version from Version space. The **updated rid** will be added to recently updated row ids of Garbage collection.

- File - function: **App.cpp - updateCommand; Column.h - addVersionVecValue**.

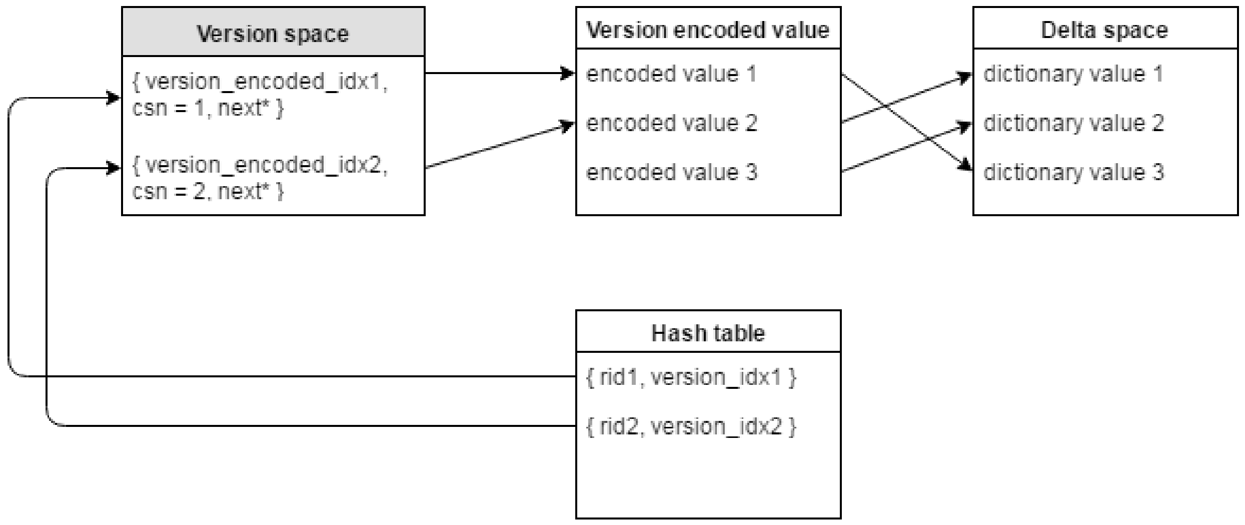


Figure 2: Column-store Version space data structure

### 3.3. Garbage Collection: GarbageCollector.h

- Garbage Collection will keep **a vector of recently updated row ids** and it can **access the list of Active transactions** from class Transaction.

- The Garbage collection will be **run as a Thread periodically** (interval = 10ms and can be modified). This thread is run by server_main.cc.

- When garbage collection runs, it will check vector of recently_updated_rowIds. If it has some values then garbage collector will run through each rid to process 2 tasks: (1) **copy the latest version** value to Data space and merge Delta space to Dictionary; (2) **remove all old versions** that not using by any transactions.

(1) The Garbage collection uses rid to **lookup in Hash table** to find the latest version in Version space vector. The dictionary value in Delta space corresponding to encoded value of latest version will be added to Dictionary (**merge Delta space and Dictionary**). The row id in Data space will be updated CSN to latest CSN.

(2) The Garbage collection will check if **history versions have CSN < start Time stamp** of all active transactions. It will traverse by next pointer through all versions and delete old version that has CSN < start time stamp. Then it **update current next pointer to NULL**. The **Hash table does not need to update** because we will not remove the latest version.

- File - function: **GarbageCollector.h - run().**

### 3.4. Scan operator:

- Command format: **SCAN|filter value|filter column|filter operator**|…

+ Filter value: value to search. As in homework 2, it will be: **56789** & **5678**

+ Filter column: column to filter. As in homework 2, it will be: **o_totalprice**.

+ Filter operator: operator to filter. It can be: =, <>, <, <=, >, >=. As in homework 2, it will be: **>, <**.

+ To add more filter we can add: filter value 2|filter column 2|filter operator 2| …

+ As in homework 2: o_totalprice > 56789 => **SCAN|56789|o_totalprice|>**

5678 < o_totalprice < 56789 => **SCAN|5678|o_totalprice|>|56789|o_totalprice|<**

- **Selection**: first, the filter value and filter condition (column & operator) will be used to filter column to find **list of result row ids**. The query process will use column's Dictionary and column's encoded value vector as descripted in Homework 2.

- **Snapshot by version**: After having a list of row ids (rid), we will project on each column to select its latest value by version (if version_flag = true). The Data space entry at index = rid will be examined. The **Scan transaction** will be created and got start time stamp (startTs).

- If this rid has **version_flag = false** then we will compare startTs with this rid's CSN. If **startTs >= CSN** then this rid data is in Transaction's result, otherwise it is not in result because of the Transaction's Start time is earlier than the Commit Time stamp of the rid. We will need to **look up on Dictionary** to find the actual data of rid's column.

- If **version_flag = true**, we will check on the **Hash table** to find an entry with **index = rid**. We will find the **pointer the latest version on Version space**.

- We will **traverse all versions** on Version space **from newest to oldest by using next pointer** to find appropriate version with **CSN <= startTs**. If no version found then there are not versions value for this Scan Transaction.

- File - function: **App.cpp - scanCommand; Column.h - selection() & projectByVersion()**

**3.5. Write conflict process:**

- The Write conflict happens when a Transaction wants to update a row id but it finds that its start time stamp (**startTs**) is **less than** row id's **CSN**. If this happens then the Transaction will be put into the Waiting list transactions to restart later.

- The **waiting transaction** will contain: **row id** to update, the **update command** and the **client** (type of ServerSocket) that produces this transaction to later send result again.

- The waiting transaction restarting process will be **a Thread that runs periodically** (for example 10 seconds).

- This Thread when running will get list of waiting transactions from Transaction class, get the update command and **execute it again**. If it can execute successfully then it return the result to client; otherwise because of still write conflict then it still in Waiting list to restart later.

- File - function: **server_main.cc - restartWaitingTransaction**.

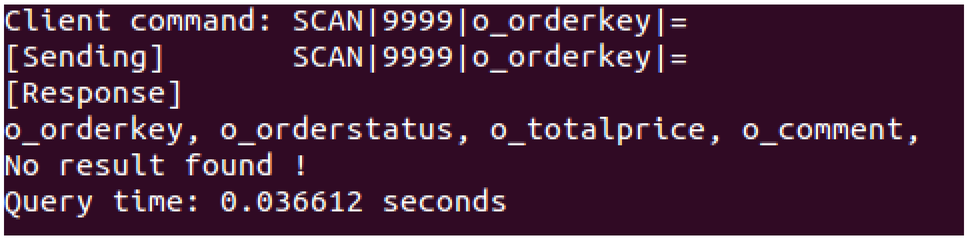
**4. Experiments**

**-** We will use a client-server system. The server runs Column-store DB with MVCC implementation. The client runs as Threads to send periodically Insert, Update, Scan commands to server to test. The client and server also logs to files (client.log & server.log) to analysis later.

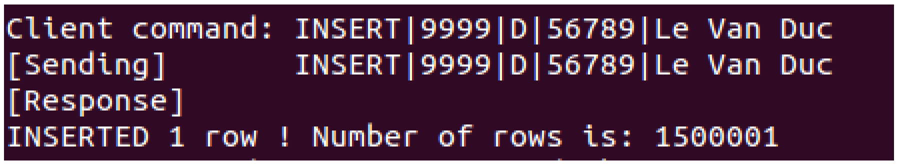
**4.1. Insert operator**

- Test scenario:

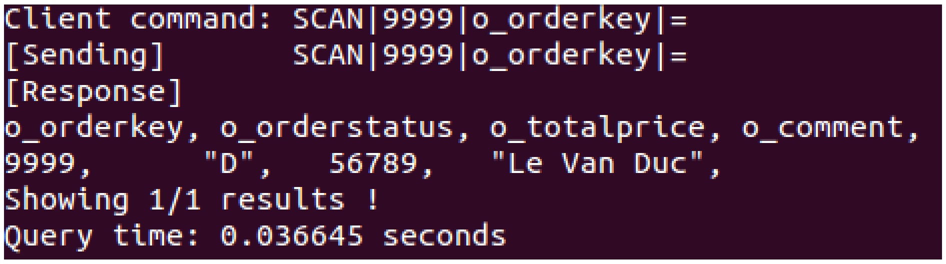
(1) SCAN|9999|o_orderkey|= : scan all o_orderkey = 9999 => **no record**


```
Client command: SCAN|9999|o_orderkey|=
[Sending]       SCAN|9999|o_orderkey|=
[Response]
o_orderkey, o_orderstatus, o_totalprice, o_comment,
No result found !
Query time: 0.036612 seconds
```

(2) INSERT|9999|D|56789|Le Van Duc : insert into orders values (9999, ‘D’, 56789, ‘Le Van Duc’) => **insert successfully** and return the row number


```
Client command: INSERT|9999|D|56789|Le Van Duc
[Sending]       INSERT|9999|D|56789|Le Van Duc
[Response]
INSERTED 1 row ! Number of rows is: 1500001
```

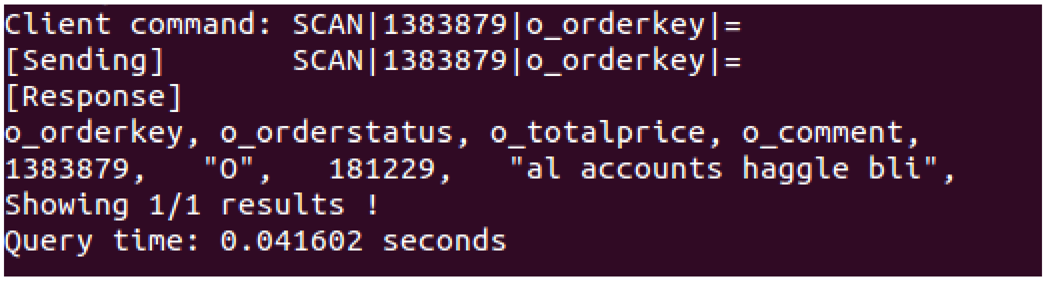
(3) SCAN|9999|o_orderkey|= : scan again with o_orderkey = 9999 => **1 record**


```
Client command: SCAN|9999|o_orderkey|=
[Sending]       SCAN|9999|o_orderkey|=
[Response]
o_orderkey, o_orderstatus, o_totalprice, o_comment,
9999,      "D",   56789,   "Le Van Duc",
Showing 1/1 results !
Query time: 0.036645 seconds
```

**4.2. Update operator: with Garbage collection and waiting list restart**

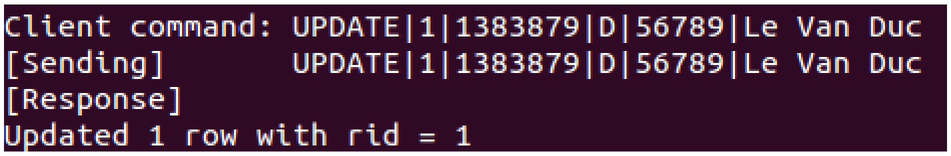
**- Test scenario 1: Update with latest version**

(1) SCAN|1383879|o_orderkey|= : scan with o_orderkey = 1383879 => **get row id = 1**



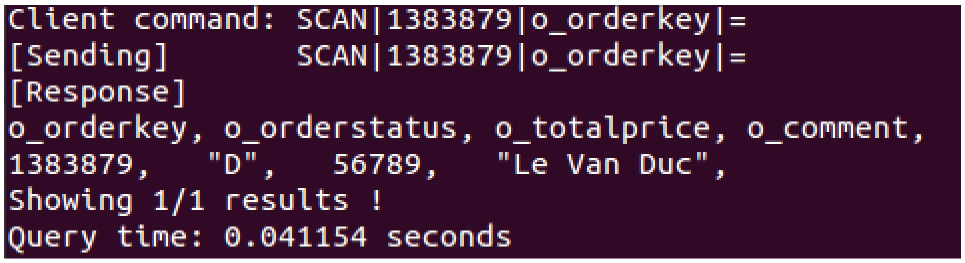
```
Client command: SCAN|1383879|o_orderkey|=
[Sending]       SCAN|1383879|o_orderkey|=
[Response]
o_orderkey, o_orderstatus, o_totalprice, o_comment,
1383879,   "O",   181229,   "al accounts haggle bli",
Showing 1/1 results !
Query time: 0.041602 seconds
```

(2) UPDATE|1|1383879|D|56789|Le Van Duc : update orders set o_orderstatus = ‘D’, o_totalprice = 56789, o_comment = ‘Le Van Duc’. Update row id = 1 => **1 row updated**.



```
Client command: UPDATE|1|1383879|D|56789|Le Van Duc
[Sending]       UPDATE|1|1383879|D|56789|Le Van Duc
[Response]
Updated 1 row with rid = 1
```

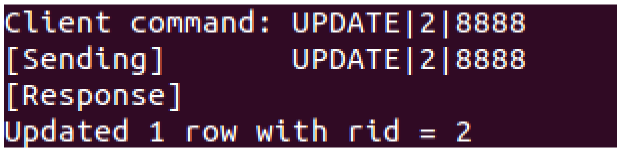
(3) SCAN|1383879|o_orderkey|= : scan again with o_orderkey = 1383879 => **get latest version** of o_orderstatus, o_totalprice, o_comment (o_orderkey no change).



```
Client command: SCAN|1383879|o_orderkey|=
[Sending]       SCAN|1383879|o_orderkey|=
[Response]
o_orderkey, o_orderstatus, o_totalprice, o_comment,
1383879,   "D",   56789,   "Le Van Duc",
Showing 1/1 results !
Query time: 0.041154 seconds
```

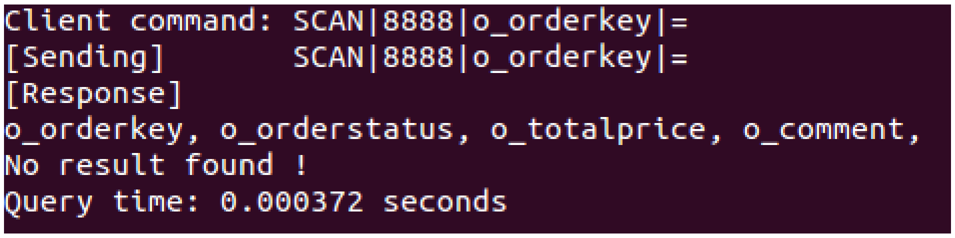
**- Test scenario 2: Update with Garbage collection**

(1) UPDATE|2|8888|o_orderkey : update orders set o_orderkey = 8888 => **1 row updated**.



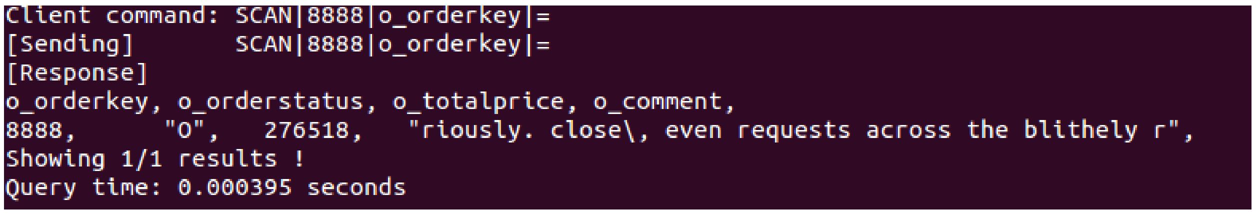
```
Client command: UPDATE|2|8888
[Sending]       UPDATE|2|8888
[Response]
Updated 1 row with rid = 2
```

(2) SCAN|8888|o_orderkey|= : scan with o_orderkey = 8888 => **no result**



```
Client command: SCAN|8888|o_orderkey|=
[Sending]       SCAN|8888|o_orderkey|=
[Response]
o_orderkey, o_orderstatus, o_totalprice, o_comment,
No result found !
Query time: 0.000372 seconds
```

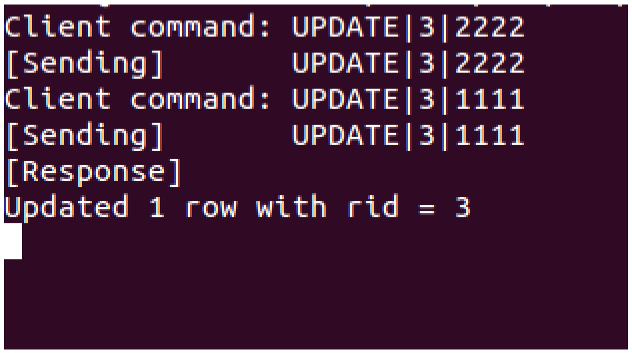
(3) **Garbage collector runs** (after about 10 seconds) => scan again => **1 record found with o_orderkey updated**:



```
Client command: SCAN|8888|o_orderkey|=
[Sending]       SCAN|8888|o_orderkey|=
[Response]
o_orderkey, o_orderstatus, o_totalprice, o_comment,
8888,      "O",   276518,   "riously. close\, even requests across the blithely r",
Showing 1/1 results !
Query time: 0.000395 seconds
```

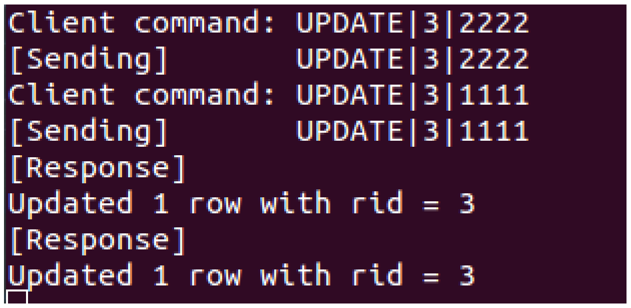
**- Test scenario 2: Update with Write conflict**

(1) UPDATE|3|1111 & UPDATE|3|2222 : update row id = 3 by 2 clients simultaneously => 1 update successfully, 1 client must **Waiting (no response)**.



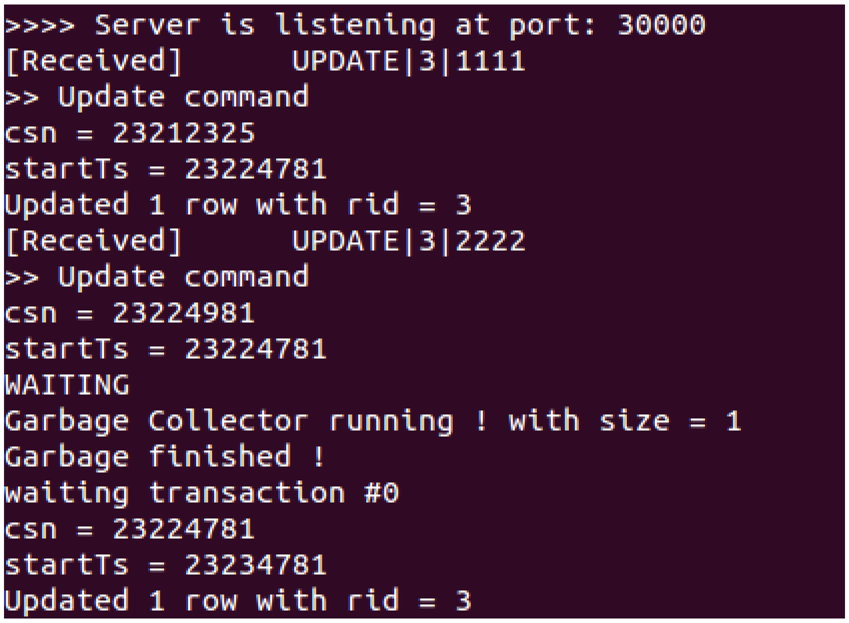
```
Client command: UPDATE|3|2222
[Sending]       UPDATE|3|2222
Client command: UPDATE|3|1111
[Sending]       UPDATE|3|1111
[Response]
Updated 1 row with rid = 3
```

(2) Restart waiting list runs => **2nd client will update successfully** (had response).



```
Client command: UPDATE|3|2222
[Sending]       UPDATE|3|2222
Client command: UPDATE|3|1111
[Sending]       UPDATE|3|1111
[Response]
Updated 1 row with rid = 3
[Response]
Updated 1 row with rid = 3
```

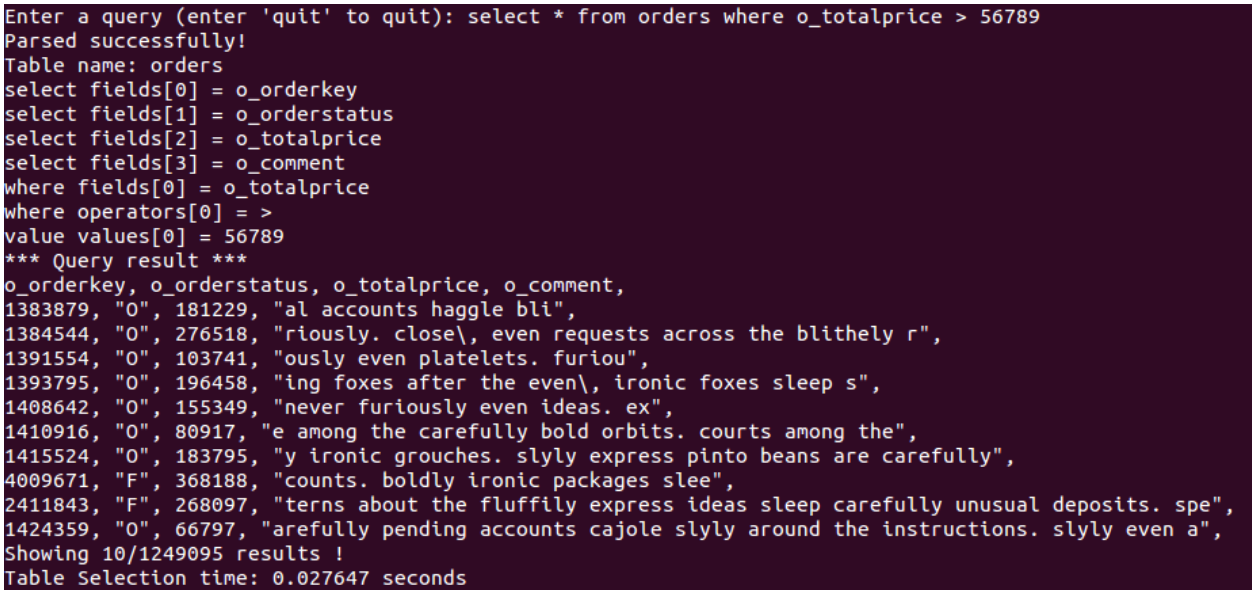
(3) Server output: show **csn & startTs**; **waiting transaction #0** has run



```
>>>> Server is listening at port: 30000
[Received]      UPDATE|3|1111
>> Update command
csn = 23212325
startTs = 23224781
Updated 1 row with rid = 3
[Received]      UPDATE|3|2222
>> Update command
csn = 23224981
startTs = 23224781
WAITING
Garbage Collector running ! with size = 1
Garbage finished !
waiting transaction #0
csn = 23224781
startTs = 23234781
Updated 1 row with rid = 3
```

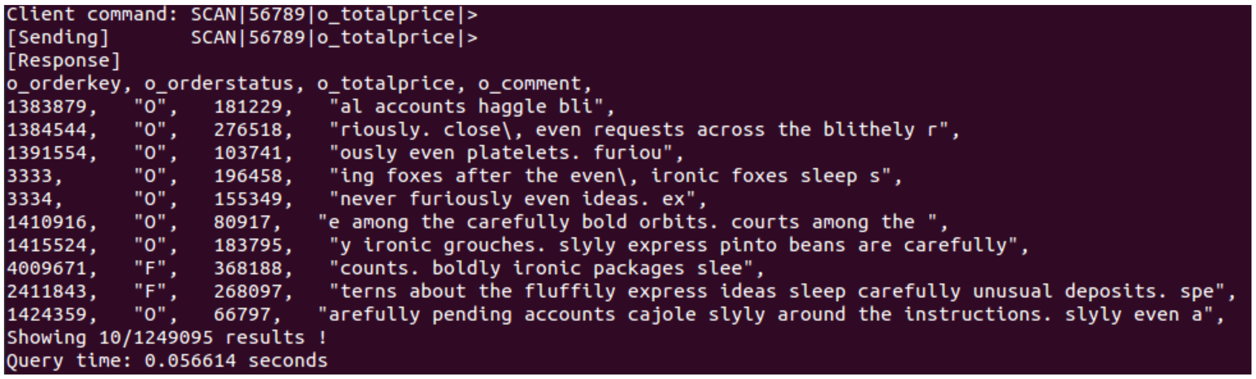
### 4.3. Scan operator: based on Homework 2 queries

(Homework 2) select * from orders where o_totalprice > 56789 => return 10 rows with 0.028s



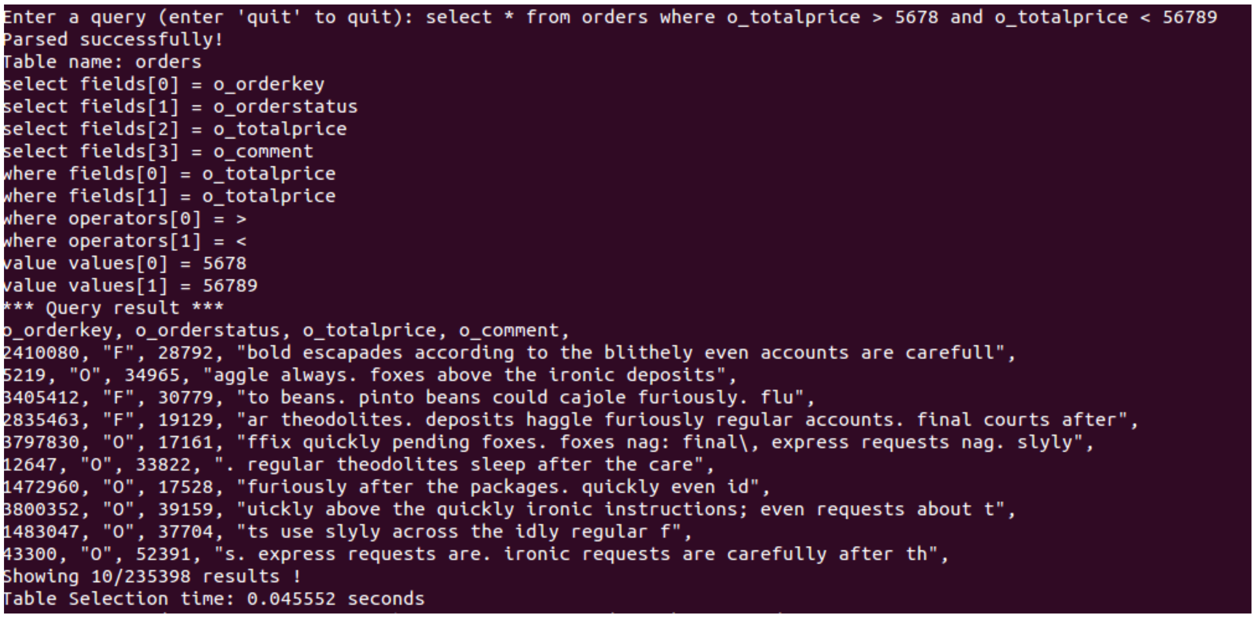
```
Enter a query (enter 'quit' to quit): select * from orders where o_totalprice > 56789
Parsed successfully!
Table name: orders
select fields[0] = o_orderkey
select fields[1] = o_orderstatus
select fields[2] = o_totalprice
select fields[3] = o_comment
where fields[0] = o_totalprice
where operators[0] = >
value values[0] = 56789
*** Query result ***
o_orderkey, o_orderstatus, o_totalprice, o_comment,
1383879, "O", 181229, "al accounts haggle bli",
1384544, "O", 276518, "riously. close\, even requests across the blithely r",
1391554, "O", 103741, "ously even platelets. furiou",
1393795, "O", 196458, "ing foxes after the even\, ironic foxes sleep s",
1408642, "O", 155349, "never furiously even ideas. ex",
1410916, "O", 80917, "e among the carefully bold orbits. courts among the",
1415524, "O", 183795, "y ironic grouches. slyly express pinto beans are carefully",
4009671, "F", 368188, "counts. boldly ironic packages slee",
2411843, "F", 268097, "terns about the fluffily express ideas sleep carefully unusual deposits. spe",
1424359, "O", 66797, "arefully pending accounts cajole slyly around the instructions. slyly even a",
Showing 10/1249095 results !
Table Selection time: 0.027647 seconds
```

(Homework 4) SCAN|56789|o_totalprice|> => return 10 rows with **0.057s**



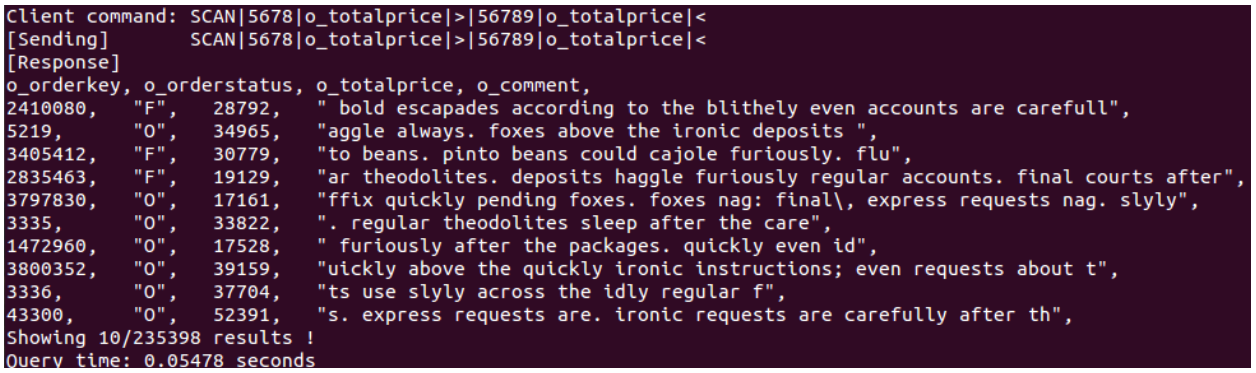
```
Client command: SCAN|56789|o_totalprice|>
[Sending]       SCAN|56789|o_totalprice|>
[Response]
o_orderkey, o_orderstatus, o_totalprice, o_comment,
1383879,   "O",   181229,   "al accounts haggle bli",
1384544,   "O",   276518,   "riously. close\, even requests across the blithely r",
1391554,   "O",   103741,   "ously even platelets. furiou",
3333,      "O",   196458,   "ing foxes after the even\, ironic foxes sleep s",
3334,      "O",   155349,   "never furiously even ideas. ex",
1410916,   "O",   80917,   "e among the carefully bold orbits. courts among the ",
1415524,   "O",   183795,   "y ironic grouches. slyly express pinto beans are carefully",
4009671,   "F",   368188,   "counts. boldly ironic packages slee",
2411843,   "F",   268097,   "terns about the fluffily express ideas sleep carefully unusual deposits. spe",
1424359,   "O",   66797,   "arefully pending accounts cajole slyly around the instructions. slyly even a",
Showing 10/1249095 results !
Query time: 0.056614 seconds
```

(Homework 2) select * from orders where o_totalprice > 5678 and o_totalprice < 56789 => get 10 rows with 0.046s



```
Enter a query (enter 'quit' to quit): select * from orders where o_totalprice > 5678 and o_totalprice < 56789
Parsed successfully!
Table name: orders
select fields[0] = o_orderkey
select fields[1] = o_orderstatus
select fields[2] = o_totalprice
select fields[3] = o_comment
where fields[0] = o_totalprice
where fields[1] = o_totalprice
where operators[0] = >
where operators[1] = <
value values[0] = 5678
value values[1] = 56789
*** Query result ***
o_orderkey, o_orderstatus, o_totalprice, o_comment,
2410080, "F", 28792, "bold escapades according to the blithely even accounts are carefull",
5219, "O", 34965, "aggle always. foxes above the ironic deposits",
3405412, "F", 30779, "to beans. pinto beans could cajole furiously. flu",
2835463, "F", 19129, "ar theodolites. deposits haggle furiously regular accounts. final courts after",
3797830, "O", 17161, "ffix quickly pending foxes. foxes nag: final\, express requests nag. slyly",
12647, "O", 33822, ". regular theodolites sleep after the care",
1472960, "O", 17528, "furiously after the packages. quickly even id",
3800352, "O", 39159, "uickly above the quickly ironic instructions; even requests about t",
1483047, "O", 37704, "ts use slyly across the idly regular f",
43300, "O", 52391, "s. express requests are. ironic requests are carefully after th",
Showing 10/235398 results !
Table Selection time: 0.045552 seconds
```

(Homework 4) SCAN|5678|o_totalprice|>|56789|o_totalprice|< => return 10 rows with **0.055s**



```
Client command: SCAN|5678|o_totalprice|>|56789|o_totalprice|<
[Sending]       SCAN|5678|o_totalprice|>|56789|o_totalprice|<
[Response]
o_orderkey, o_orderstatus, o_totalprice, o_comment,
2410080,   "F",   28792,   " bold escapades according to the blithely even accounts are carefull",
5219,      "O",   34965,   "aggle always. foxes above the ironic deposits ",
3405412,   "F",   30779,   "to beans. pinto beans could cajole furiously. flu",
2835463,   "F",   19129,   "ar theodolites. deposits haggle furiously regular accounts. final courts after",
3797830,   "O",   17161,   "ffix quickly pending foxes. foxes nag: final\, express requests nag. slyly",
3335,      "O",   33822,   ". regular theodolites sleep after the care",
1472960,   "O",   17528,   " furiously after the packages. quickly even id",
3800352,   "O",   39159,   "uickly above the quickly ironic instructions; even requests about t",
3336,      "O",   37704,   "ts use slyly across the idly regular f",
43300,     "O",   52391,   "s. express requests are. ironic requests are carefully after th",
Showing 10/235398 results !
Query time: 0.05478 seconds
```

- Comment: The execution time with MVCC implementation is **not much greater than no MVCC** but this is the result with just a few update, not heavily update.